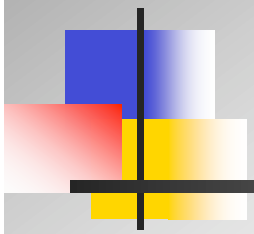
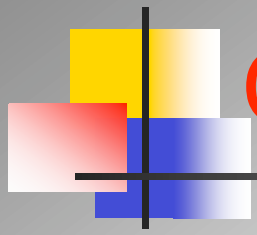


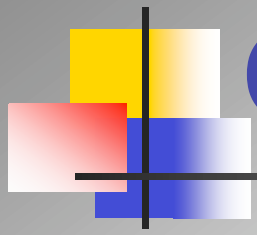
Benchmarks on BG/L: Parallel and Serial

John A. Gunnels
Mathematical Sciences Dept.
IBM T. J. Watson Research Center



Overview

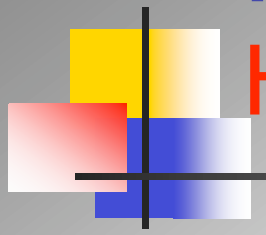
- Single node experience
  - Architectural impact
  - Algorithms
- Linpack
  - Dealing with a bottleneck
  - Communication operations



Compute Node: BG/L

- Dual FPU/SIMD
  - Alignment issues
- Three-level cache
  - Pre-fetching
- Dual Core
- Non-coherent L1 caches
  - 32 KB, 64-way, Round-Robin
  - L2 & L3 caches coherent

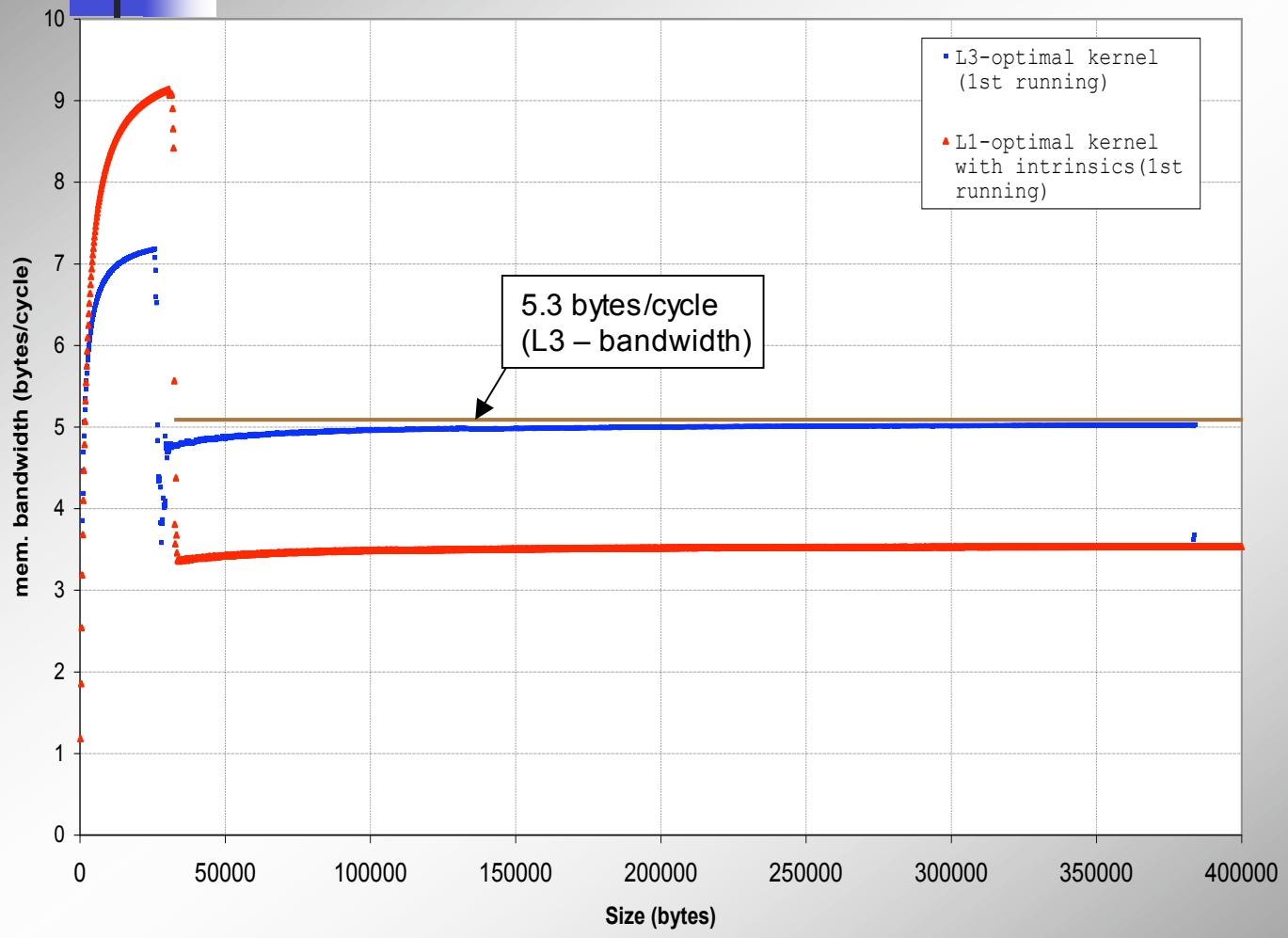
Programming Options
High → Low Level



- Compiler optimization to find SIMD parallelism
  - User input for specifying memory alignment and lack of aliasing
    - `alignx` assertion
    - `disjoint` pragma
- Dual FPU intrinsics ("built-ins")
  - Complex data type used to model pair of double-precision numbers that occupy a (P, S) register pair
  - Compiler responsible for register allocation and scheduling
- In-line assembly
  - User responsible for instruction selection, register allocation, and scheduling

L1 and L3-optimal DGEMV Bandwidth Utilization

Memory Bandwidth Utilization for L3-optimal DGEMV kernel



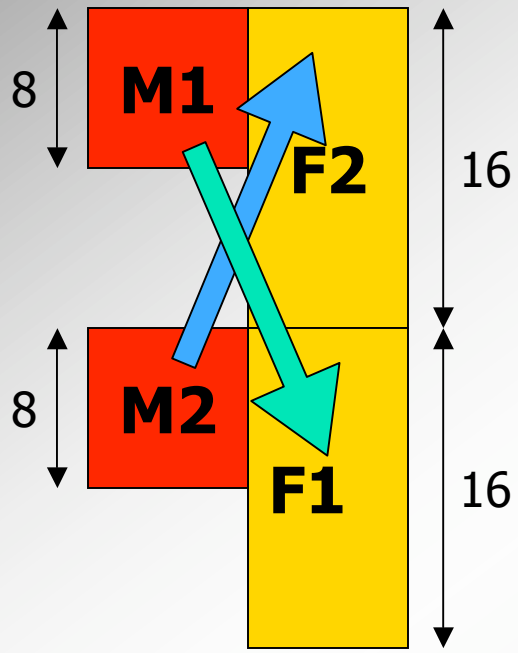
Two different kernels are needed to deal with data when:

-Data come out of L1

-Data come out of L3

Matrix Multiplication

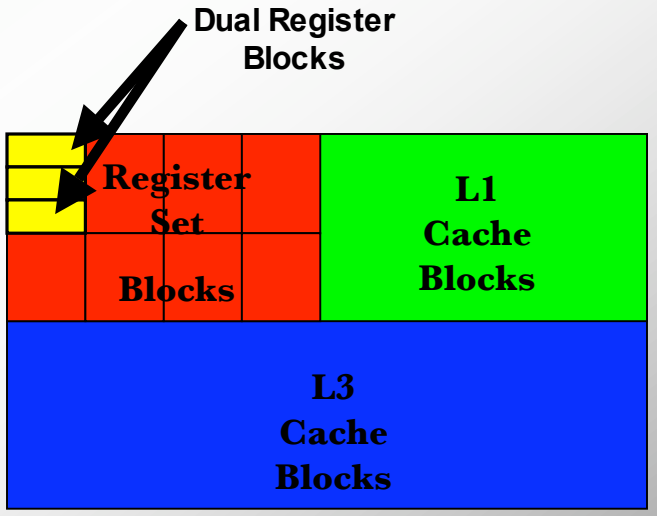
Tiling for Registers (Analysis)



- Latency tolerance (not bandwidth)
  - Take advantage of register count
- Unroll by factor of two
  - 24 register pairs
  - 32 cycles per unrolled iteration
  - 15 cycle load-to-use latency (L2 hit)
- Could go to 3-way unroll if needed
  - 32 register pairs
  - 32 cycles per unrolled iteration
  - 31 cycle load-to-use latency

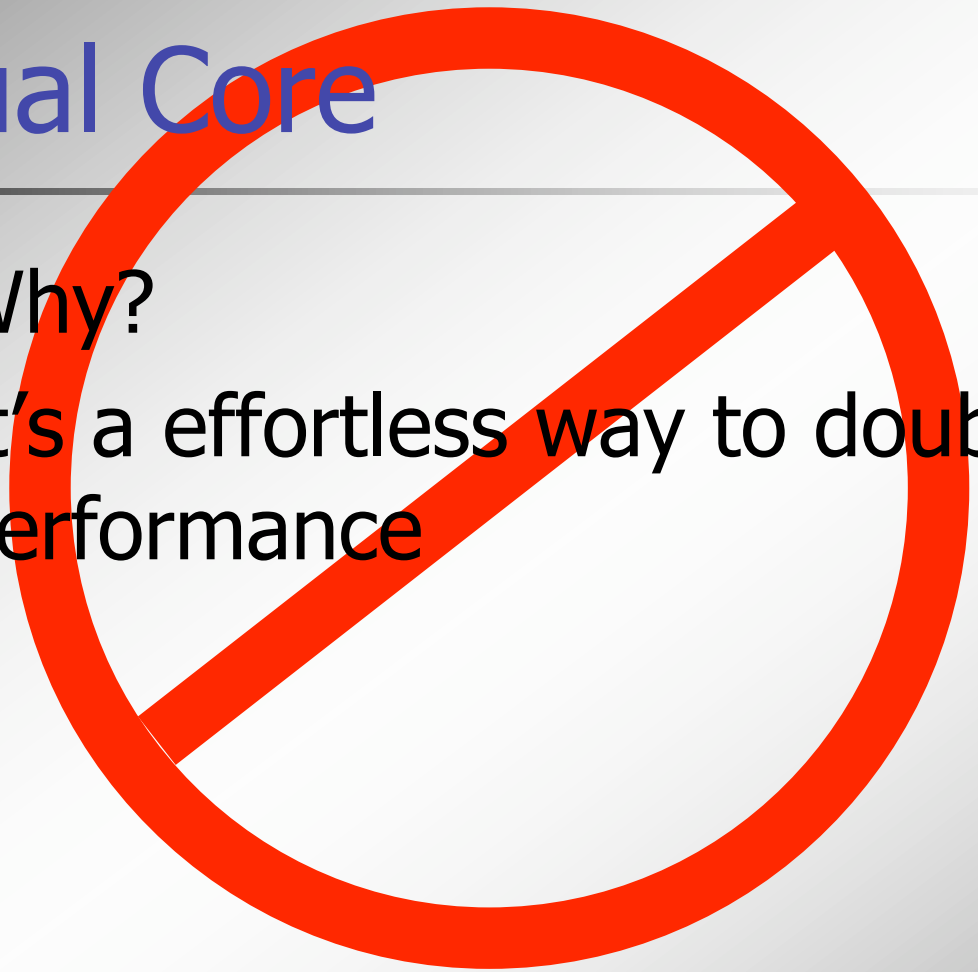
Recursive Data Format

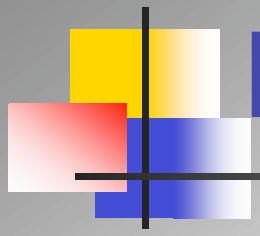
- Mapping 2-D (Matrix) to 1-D (RAM)
  - C/Fortran do not map well
- Space-Filling Curve Approximation
  - Recursive **Tiling**
- Enables
  - Streaming/pre-fetching
  - **Dual core** "scaling"



Dual Core

- Why?
- It’s a effortless way to double your performance

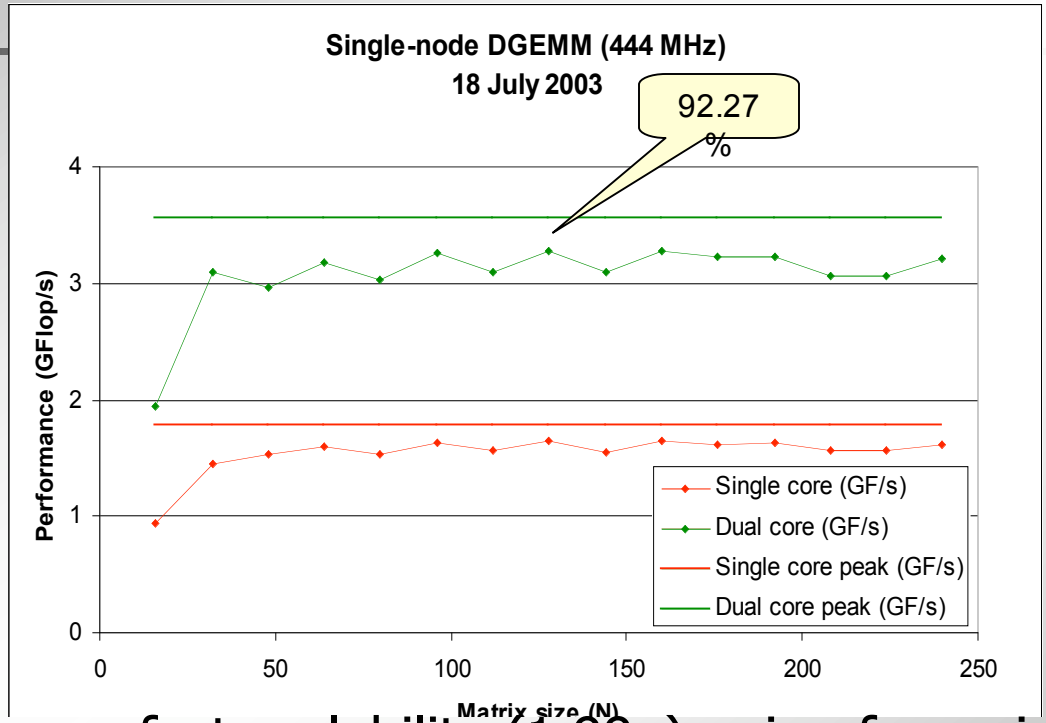




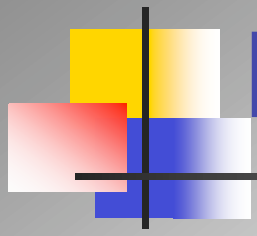
Dual Core

- Why?
- It exploits the architecture and may allow one to double the performance of their code in some cases/regions

Single-Node DGEMM Performance at 92% of Peak

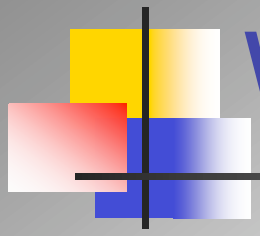


- Near-perfect scalability (1.99×) going from single-core to dual-core
- Dual-core code delivers 92.27% of peak flops (8 flop/pclk)
- Performance (as fraction of peak) competitive with that of Power3 and Power4



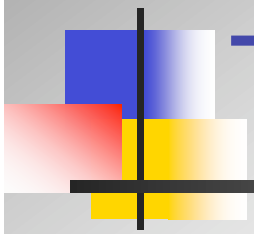
Points to consider

- Code fusion can enable one to
  - Perform a data re-format and/or make effective use of both cores for an operation
- The architecture is very rich
  - Corner cases have to be handled
  - Can be very powerful
  - Helpful in understanding performance
  - Semi-esoteric improvements exist
    - Fine-grained L1 data cache control



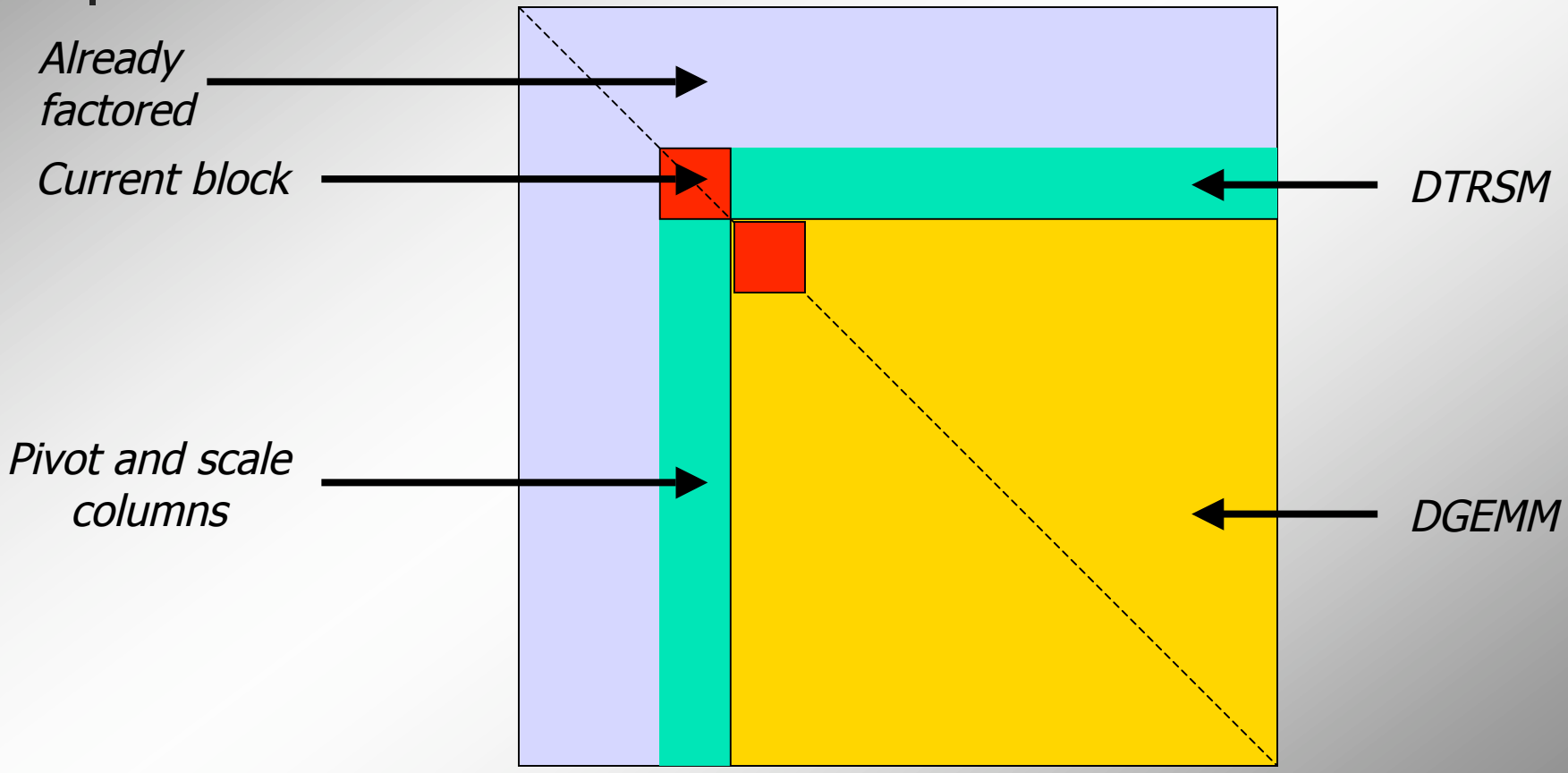
What More Could We Want?

- Open up the cache architecture more
  - It would be good if the library writer could specify that a particular access would be a miss in L1, or a hit in L3, for example
  - Expose more microarchitectural constraints to the compiler
  - Example: maximum number of L1 cache misses before stall
- Better register scheduling algorithms
  - Currently, we have observed excessive spills when using close to all 32 registers

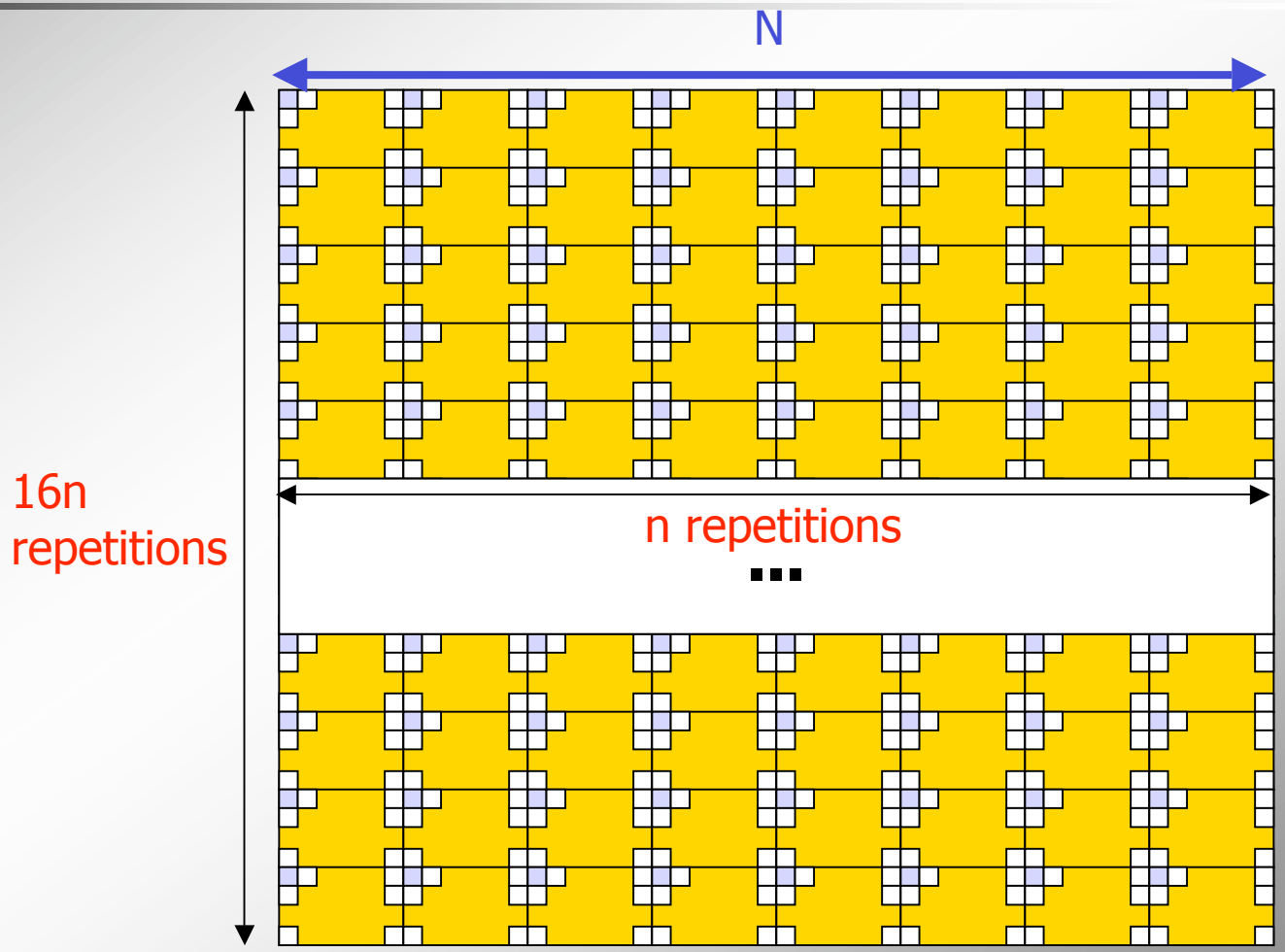


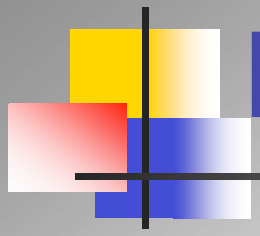
The Linpack Benchmark

LU Factorization: Brief Review



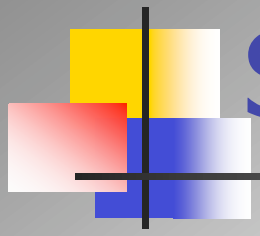
LINPACK
Problem Mapping





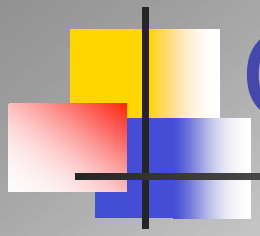
Panel Factorization: Option #1

- **Stagger the computations**
- PF Distributed over relatively few processors
- May take as long as several DGEMM updates
- DGEMM load imbalance
  - Block size trades balance for speed
- Use collective communication primitives
  - May require no “holes” in communication fabric



Speed-up Option #2

- **Change the data distribution**
  - Decrease the critical path length
  - Consider the communication abilities of machine
- Complements Option #1
- Memory size (small favors #2; large #1)
  - Memory hierarchy (higher latency: #1)
- The two options can be used in concert

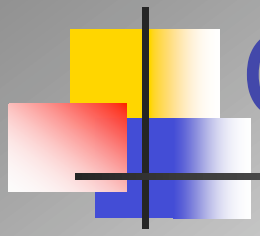


Communication Routines

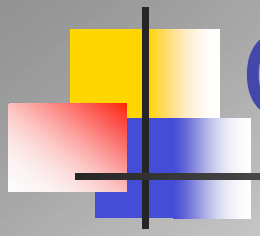
- Broadcasts precede DGEMM update
- Needs to be architecturally aware
  - Multiple “pipes” connect processors
- Physical to logical mapping
- Careful orchestration is required to take advantage of machines considerable abilities
- See: MPI Presentation (MPI_Bcast)

What Else?

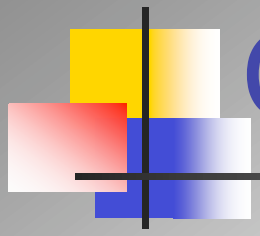
- **It's a(n) …**
  - FPU Test
  - Memory Test
  - Power Test
  - Torus Test
  - Mode Test (Virtual/Co-)



Conclusion: Scaling

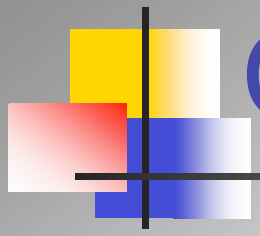


Conclusion: Scaling



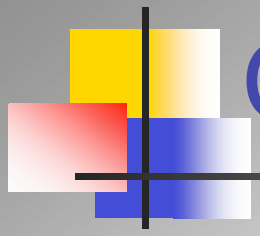
Conclusion: Scaling

- Contributions to lack of “flat” scaling
  - Time spent tuning for a particular configuration
  - Different driver versions evidence different characteristics
    - Runs performed at different stages
  - Physical layout of machine
    - Aspect ratio



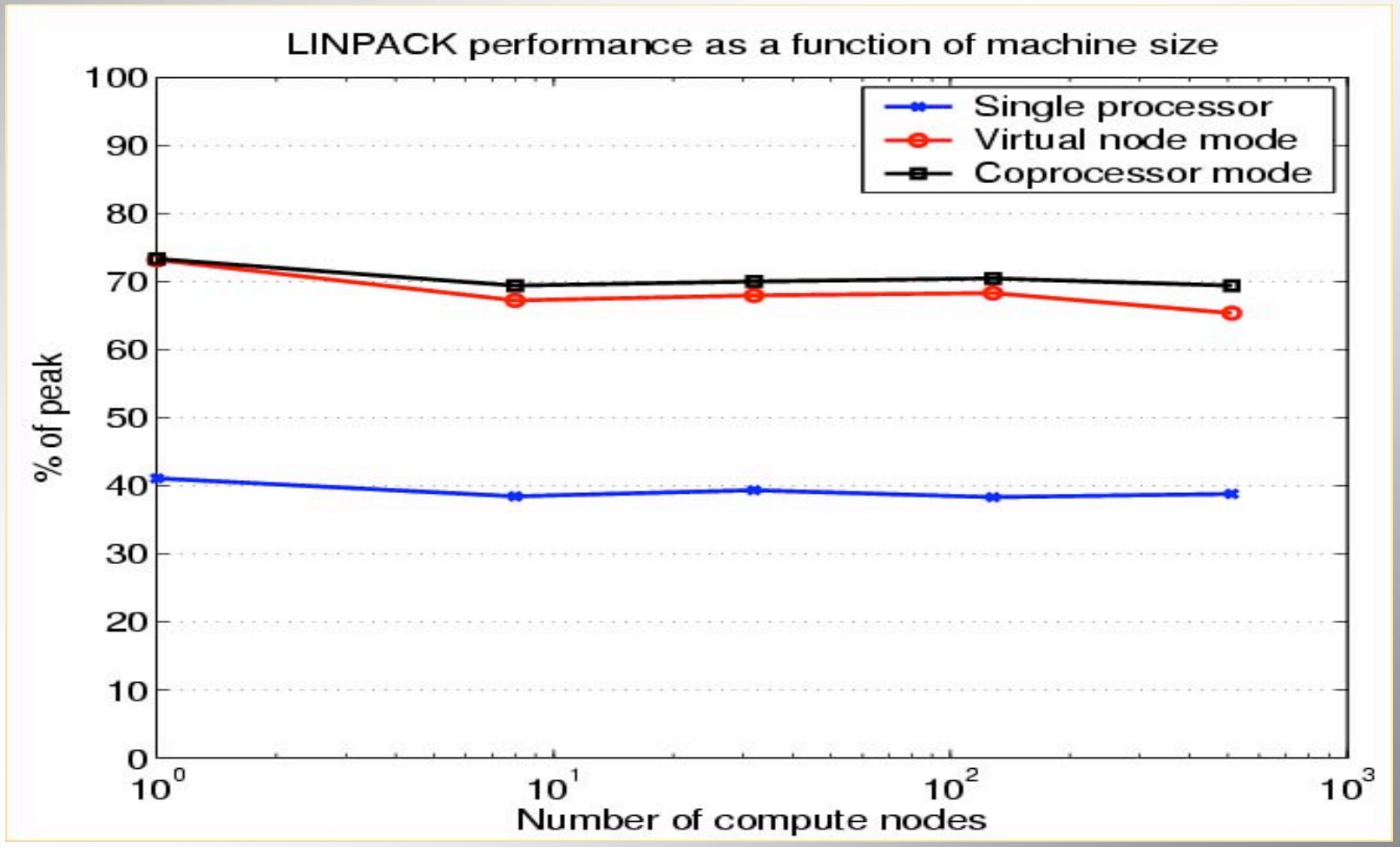
Conclusion

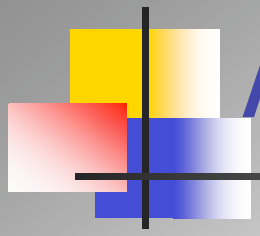
- #73 in TOP500 List (11/2003)
  - Limited Machine Access Time
    - Made analysis/model more important
- #4 (4096 DD1) & #8 (2048 DD2) on 6/2004 TOP500
- **#1 on 11/2004 TOP500**
  - Also:#8 (4096 DD1) & #15 (2048 DD2)



Conclusion: **Breakdown** (old data)

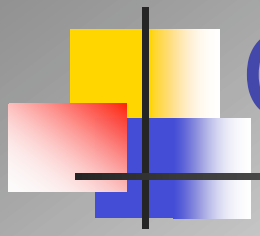
What about VNM?





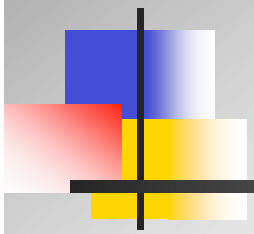
Additional Conclusions

- Models, extrapolated data
  - Use models to the extent that the architecture and algorithm are understood
  - Extrapolate from small processor sets
  - Vary as many (yes) parameters as possible at the same time
    - Consider how they interact and how they don't
    - Also remember that instruments affect timing
      - Often can compensate (incorrect answer results)
  - Utilize observed "eccentricities" with caution (MPI_Reduce)



Current Fronts

- HPC Challenge Benchmark Suite
  - STREAMS, HPL, etc.
- HPCS Productivity Benchmarks
- Math Libraries
- Focused Feedback to Toronto
- PERCS Compiler/Persistent Optimization
- Linpack Algorithm on Other Machines



Benchmarks on BG/L: Parallel and Serial

John A. Gunnels
Mathematical Sciences Dept.
IBM T. J. Watson Research Center